VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Inteligentní bot pro hru StarCraft: Brood War

Intelligent Bot for the Game StarCraft: Brood War

2015 Lubomír Sikora

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lubomír Sikora**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Inteligentní bot pro hru StarCraft: Brood War
Intelligent Bot for the Game StarCraft: Brood War

Jazyk vypracování: čeština

Zásady pro vypracování:

Hlavním cílem bude naprogramování bota (počítačového hráče zastupujícího člověka, zkrácenina slova robot) do strategické hry StarCraft: Brood War. Bot by měl umět inteligentně reagovat na podněty nepřítele, popř. se učit vlastními prohrami, adaptivně měnit své strategie s cílem maximalizace budoucích úspěchů. Pro realizaci se bude používat aplikační rozhraní BWAPI a jeho interface JNIBWAPI pro jazyk Java. Je to nejjednodušší způsob, jak programovat AI na zmiňovanou hru. Hra StarCraft je velmi známá starší strategická hra, kde má hráč na výběr ze tří ras. Každá z nich má svoje unikátní technologie i způsob hraní. Hráč buduje svou základnu a armádu. Jeho cílem je zničit základnu nepřítele.
Odkaz na hru: http://eu.blizzard.com/en-gb/games/sc/
Odkaz na BWAPI: https://code.google.com/p/bwapi/

1. Zhodnoťte současný stav na poli botů a počítačových her.
2. Stanovte oblasti, kde se tento přístup využívá a kde je zatím nevyužit, příp. málo využíván.
3. Navrhněte vlastní řešení vybraného problému.
4. Vytvořte program a otestujte jej.
5. Zhodnoťte výsledky v závěru.

Seznam doporučené odborné literatury:

[1] Mark Watson, Practical Artificial Intelligence Programming With Java, November 11, 2008
[2] Edwin Wise, Hands-on AI with Java : Smart Gaming, Robots, and More, McGraw-Hill/TAB Electronics, ISBN: 978-0071424967, 2004
[3] Joseph P. Bigus, Constructing Intelligent Agents Using Java: Professional Developer's Guide, Wiley, ISBN 978-0471396017
[4] Mat Buckland, Programming Game AI By Example (Wordware Game Developers Library), Jones & Bartlett Learning, ISBN 978-1556220784
[5] Jason Gregory, Game Engine Architecture, A K Peters/CRC Press, ISBN 978-1568814131
[6] David Pallmann, Programing Bots, Spiders, and Intelligent Agents in Microsoft Visual C++, Microsoft press, ISBN 978-0735605657
[7] Ian Millington, Artificial Intelligence for Games, CRC Press, ISBN 978-0123747310

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **prof. Ing. Ivan Zelinka, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Tímto chci poděkovat prof. Ing. Ivanu Zelinkovi Ph.D., jakožto vedoucímu práce, za rady a odbornou pomoc, kterou mi vždy neváhal poskytnout, vstřícný přístup a kvalitní odborné vedení, kterého se mi od něj dostávalo.

Abstrakt

Práce je soustředěna na techniky umělé inteligence a na jejich praktické použití. Cílem práce je naprogramovat počítačového hráče nahrazujícího člověka do strategické hry StarCraft: Brood War. Implementace využívá klasických technik z prostředí umělé inteligence, stejně jako se snaží využít nekonvenčních technik, jakými jsou evoluční výpočetní techniky. Počítačový hráč je řešen implementací rozhodovacího stromu, společně s evolučním algoritmem SOMA. Vše bylo psané v programovacím jazyce Java. V implementaci umělé inteligence jsem vytvořil systém, který jednoduchým způsobem zajišťuje chování počítačového hráče. Moje konkrétní implementace algoritmu SOMA poskytuje možnosti efektivního, koordinovaného pohybu bojových jednotek po mapě. Práce ukázala velký přínos evolučních výpočetních technik na poli strategických her.

Klíčová slova: umělá inteligence, rozhodovací algoritmus, evoluční výpočetní techniky, evoluční algoritmus

Abstract

This work is focused on techniques of artificial intelligence and on their practical utilization. The goal of the work is to implement computer player replacing human in real time strategy StarCraft: Brood War. The implementation uses conventional techniques from scope of artificial intelligence, as it at same time endeavors use of unconventional techniques, such as evolutionary computation. The computer player is provided by implementation of decision-making tree together with evolutionary algorithm called SOMA. Everything was written in programming language Java. I created system, which ensures behavior of computer player in an easy way in implementation of artificial intelligence. My particular implementation of SOMA algorithm provides an opportunity for efficient, coordinated movement of combat units over the map. The work has shown great benefit of evolutionary techniques in the field of real time strategy games.

Keywords: artificial intelligence, decision making algorithm, evolutionary computing, evolutionary algorithm

Seznam použitých zkratek a symbolů

ACO	–	algoritmus Ant Colony Optimalization
EA	–	evoluční algoritmy
EVT	–	voluční výpočetní techniky
rnd	–	náhodné číslo
SOMA	–	algoritmus Self Organising Migration Algorithm

Obsah

1 Úvod **5**

2 Umělá inteligence **6**
2.1 Co je umělá inteligence 6
2.2 Bot 7
2.3 Zhodnocení stavu umělé inteligence na poli her 7

3 Rozhodovací techniky **9**
3.1 Rozhodovací strom 9
3.2 Stavový automat 11
3.3 Cílově orientované chování 13
3.4 Systém založený na pravidlech 14
3.5 Závěr kapitoly 15

4 Evoluční výpočetní techniky **16**
4.1 Obecný princip 16
4.2 SOMA – Self organising migration algorithm 19
4.3 Princip algoritmu SOMA 23

5 Seznámení s hrou **26**
5.1 Popis hry 26

6 Implementace **30**
6.1 Pohyb 31
6.2 Simulace 36

7 Závěr **38**

8 Reference **39**

Přílohy **40**

Seznam tabulek

1 Význam parametrů SOMA Zdroj: [10]. 20
2 Příklad perturbačního vektoru pro jedince o 4 parametrech a $PRT = 0,3$. 22
3 Význam biologické terminologie v algoritmu SOMA, Zdroj: [10] 23
4 Parametry algoritmu SOMA 31
5 Nastavení vah 33
6 Skóre simulací 37

Seznam obrázků

1 rozhodovací strom 9
2 rozšířená struktura rozhodovacího stromu 10
3 stavový automat 11
4 deterministický stavový automat 12
5 Obecný cyklus evolučního algoritmu. 16
6 Křížení v klasických genetických algoritmů 18
7 Příklad mutace 18
8 Princip ukončovacího parametru *MinDiv*, Zdroj: [10] 20
9 Princip SOMA, Zdroj: [10] 24
10 Protoská zásobovací budova: Pylon 27
11 Terranská zásobovací budova: Supply Depot 27
12 Zergská zásobovací jednotka: Overlord 27
13 Protoská hlavní budova dělnická jednotka: Probe 28
14 Terranská hlavní budova dělnická jednotka: SCV 28
15 Zergská hlavní budova dělnická jednotka: Drone 29
16 Skupinka Zerglingů 36
17 Skupinka Zealotů 37
18 Hydraliskové vlevo a Lurkeři vpravo 37
19 Protoská základna 1 41
20 Protoská základna 2 41
21 Terranská základna 1 42
22 Terranská základna 2 42
23 Zergská základna 1 43
24 Zergská základna 2 43
25 Protoss napadený Terranem (dominující letecká jednotka se jmenuje Battlecruiser) 44
26 Ukázka síly Protosů (průhledné jednotky jsou neviditelné) 44
27 Terranská vzdušná jednotka Battlecruiser používá schopnost Yamato cannon 45
28 Protoské kouzlo Psionic Storm proti mase zergských jednotek 45
29 některé zergské jednotky (zleva: Ultralisk, Queen, Defiler; nahoře Overlord) 46
30 Zergské vzdušné síly (zleva: Mutalisk, Guardian, Devourer; nahoře: Scourge) 46

Seznam příloh

Příloha A Obrázky ze hry 41
Příloha A.1 Protoská základna 41
Příloha A.2 Terranská základna 42
Příloha A.3 Zergská základna 43
Příloha A.4 Další obrázky ze hry 44

Příloha B Příloha na CD/DVD 47

1 Úvod

Obsah této práce je zaměřen na techniky spojené s problematikou umělé inteligence a jejich použití. Dává pojem o tom, co za techniky se používají v umělé inteligenci i dnes. Popisuje problematiku spojenou s rozhodováním, dále problematikou evolučních výpočetních technik a jejich vhodnosti, co se týče použití v této oblasti. Umělá inteligence je v dnešní době použitá v několika oblastech. Jimi jsou např. počítačové hry, kdy umělá inteligence zastupuje lidského hráče. Dále je třeba umělá inteligence použitá v osobních asistentech. Do této kategorie spadají asistenti jako Siri (Apple), Cortana (Microsoft) a další. V této práci se budu věnovat převážně umělé inteligenci spojené s počítačovými hrami.

Ten kdo čte tuto práci, se jistě setkal s nějakou počítačovou hrou. Ať už to bylo z doslechu, nebo přímo jako hráč. Ta druhá skupina lidí, tedy hráčů, mi jistě dá za pravdu, že by si většinu her bez kvalitní umělé inteligence jen málo užili. Toto téma jsem si vybral z důvodu, že je v této oblasti nekonečný prostor pro rozvoj. Dále proto, že sám jsem, nebo jsem byl hráčem počítačových her a vždycky mě zajímalo, jak to vše vlastně funguje. Taky jsem věděl, že tyto techniky nejsou dobré pouze pro aplikace umělé inteligence, ať už ve své herní podobě, nebo podobě chování inteligentních robotů, které si většinou spojíme s národem vycházejícího slunce. Tyto techniky jsou použitelné ve více oborech informačních technologií, převážně pak v metodách analýzy dat. Z těchto důvodů nemůže být téma aktuálnější. Umělá inteligence je deviza moderní doby. Hráčů počítačových her ubývat nebude a budou požadovat stále kvalitnější a kvalitnější požitek ze hry.

Mým úkolem v této práci bylo prakticky implementovat počítačového hráče, neboli bota, do strategické hry StarCraft: Broot War. Cílem tedy bylo nejen si vyzkoušet standardní techniky spojené s touto problematikou ale také využít nekonvenčních technik a otestovat jejich vhodnost v tomto konkrétním příkladu. Nekonvenčními technikami jsou myšleny evoluční výpočetní techniky, kterým se budu věnovat v teoretické části práce.

Literatur na téma umělé inteligence je mnoho. Ohledně inteligentních agentů se vyjadřuje [1, 6, 7]. Herní umělou inteligencí se zabývá třeba [2, 5, 9], kterým tato práce dává svůj prostor. Existuje mnoho publikací, které se zabývají praktickou stránkou věci. Jmenovitě to jsou třeba [8, 2].

2 Umělá inteligence

2.1 Co je umělá inteligence

„Umělá inteligence je o uzpůsobení počítačů k provádění myšlenkové činnosti, které jsou schopni lidé a zvířata.“ [1] [5] V některých věcech počítače excelují. Dokáží vyřešit různé problémy matematické, vyhledávací, třídicí, a jiné. Jsou nepřekonatelné v problémech, které lze vyřešit algoritmicky, tedy jasně daným postupem. Máme jasně daný matematický postup k vypočítání vzdálenosti mezi dvěma body, známe postup k vypočtení diskriminantu, je znám postup k nalezení nejmenší hodnoty mezi čísly atd. Počítače taky dokáží hrát některé deskové hry lépe, než člověk. Počítač společnosti IBM jménem „Deep Blue“ poprvé porazil šachového velmistra Garri Kasparova dne 10. 2. 1996. [5] tvrdí, že tyto problémy zprvu byly považovány za problémy umělé inteligence, ale protože postupem času byly vyřešeny stále obsáhlejšími způsoby, byly vyřazeny z oblasti umělé inteligence. Naopak existují problémy, ve kterých počítače nejsou dobré, ale člověku přijdou triviální. Jsou to problémy jako řeč, rozpoznávání obličejů, rozhodování, kreativita. Tyto problémy se tedy řadí do domény umělé inteligence. [2] rozděluje umělou inteligenci do dvou sfér. Jedna je akademická, a druhá je umělá inteligence použitá ve hrách. Ta akademická se ještě dělí na dvě podkategorie. Silnou umělou inteligenci (strong AI) a slabou umělou inteligenci (weak AI). Silná umělá inteligence se snaží napodobit lidský myšlenkový proces, a slabá umělá inteligence aplikuje technologie umělé inteligence na řešení problémů reálného světa. Oba tyto typy mají ovšem něco společného. Snaží se vyřešit problém optimálně. Vědci nechávají běžet superpočítače o tisících jádrech hodiny, dny, nebo i týdny. To vše jen pro to, aby dostali optimálního výsledku. Na druhé straně herní umělá inteligence nemá možnosti takovýchto výpočetních zdrojů a musí tedy volit kompromis mezi ideálností výsledku a délkou výpočetního času. Ta se liší stroj od stroje, na kterém hra poběží. Nejdůležitějším cílem v herní umělé inteligenci je uspokojit a bavit hráče. V herní umělé inteligenci ani není nutné vyhledat optimální řešení. Představte si sebe, jakožto hráče počítačových her a v jakékoli hře, kterou hrajete, prohrajete, protože byl nepřítel příliš chytrý. Od takovéto hry byste velmi rychle upustili. Avšak v zájmu herních společností je scénář opačný. Chtějí, aby vás hra bavila co možná nejdéle. Na druhou stranu by měly hry představovat lehkou výzvu, aby hráče nenudily. Takže není ani chtěné udělat nepřítele příliš hloupého.

Umělá inteligence, tedy studuje syntézu a analýzu výpočetních agentů, kteří se chovají inteligentně, tvrdí [7]. Agentem se myslí cokoli, co se nějak chová. Může to být pes, letadlo, termostat, robot, člověk, firma atd.

[1] Překlad z originálu: Artificial intelligence is about making computers able to perform the thinking tasks that humans and animals are capable of.

2.2 Bot

Ve spojitosti s hrami se velice často setkáváme s pojmem bot. Bot je hráč ovládaný počítačem, který se snaží napodobit chování lidského hráče. Je to také zkrácenina slova robot. S tímto pojmem se setkáváme, pokud hrajeme hru, která je určená pro více hráčů, ale nemáme tu možnost obsadit všechny pozice lidskými zdroji. Většinou není těžké rozpoznat bota od lidského hráče. Je to hlavně z toho důvodu, že lidské myšlení je natolik složitý proces, že se nedá naprogramovat. Dnešní boti jsou však často velice chytří a sofistikovaní hráči. Ve hrách máme většinou i tu možnost, nastavit úroveň obtížnosti bota a ten bude poté dle nastavení chytřejší. S boty se můžeme setkat ve hrách jako StarCraft [Blizzard Entertainment, 1998], série Heroes of Might and Magic [Ubisoft Entertainment], série Need for Speed [Electronic Arts] a další.

S pojmem bot se také můžeme setkat v prostředí internetu. *Webový Bot* se typicky používá pro indexování webu. [6] říká, že weboví boti jsou také nazýváni webovými pavouky nebo webovými červy.

2.3 Zhodnocení stavu umělé inteligence na poli her

V dnešní době je na trhu nepřeberné množství her. Počínaje sportovními hrami, simulátory, přes dobrodružné arkádové hry, hry, v nichž si hrajeme na hrdiny (RPG – roleplaying games), strategické hry a další hry, které spadají do kategorie různých žánrů. Hru může hrát jak jen jeden hráč (singleplayer), tak i více hráčů (multiplayer), buď u jednoho počítače, takovéto hry jsou vesměs tahové, nebo hry hrající se masivně přes internet (MMO – massively multiplayer online). Věc, kterou ale mají společnou je potřeba umělé inteligence. Ale mají všechny hry umělou inteligenci na vysoké úrovni?

Podívejme se do minulosti. Hra Pacman [Midway Games West, Inc., 1979] dávala hráči pocit, že nepřátelé nějakým způsobem reagují na váš pohyb. Dokázali vás pronásledovat. Zde byla využita technika zvaná „stavový automat“ tvrdí [5]. Duchové, jak se tamním monstrům začalo říkat, měli dva stavy. Jeden byl únikový, druhý pronásledovací. Pokud se duch nacházel ve stavu pronásledovacím, pronásledoval hráče způsobem, že podle vygenerovaného pseudonáhodného čísla na každé křižovatce buď pokračoval ve sledování, nebo se vydal jiným náhodným směrem. V únikovém stavu se prostě generovala náhodná trasa na každé křižovatce opět podle pseudonáhodného čísla. Vše bylo velice jednoduché. Ale od roku 1979 se toho změnilo. Jako další stojí za zmínku RTS (real time strategy) Warcraft [Blizzard Entertainment, 1994], který v sobě měl zabudovaný systém hledání cesty (pathfinding system). Postupem času se umělá inteligence začala stávat složitější a nabývat na komplexitě. V roce 2001 hra „Black And While“ [LionheadStudiosLtd. 2001] měla implementovanou neuronovou síť[2] v mozku každé kreatury. Avšak někdy se tyto stvoření chovaly velice hloupě, tvrdí [5].

Jak je vidět tak umělá inteligence procházela vývojem od samého počátku po přítomnost. Je to ten obor, který se může zdokonalovat do nekonečna. Umělé inteligence v některých hrách však mohou vypadat hloupě na vzory známým technikám v době jejich vydání. Například hra Grand Theft Auto: San Andreas [Rockstar Games, 2004]. Někdy

[2]Technologie umělé inteligence, která se snaží napodobit činnost neuronů v mozku

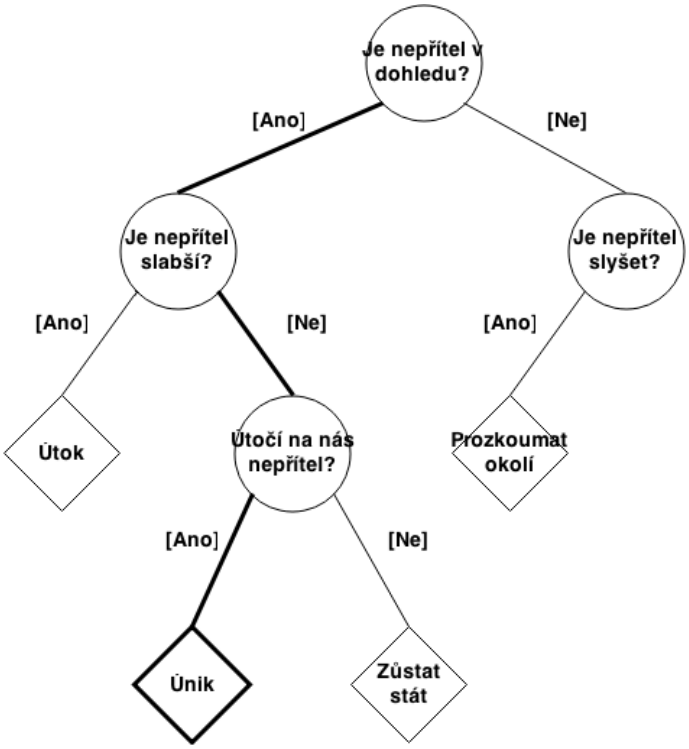
se npc (non-player character - charakter, kterým není hráč) chovaly extrémně hloupě. Řidiči sanitek a hasiči byli svému okolí spíše nebezpeční. Dále třeba hra StarCraft [Blizzard Entertainment, 1998]. Bojové jednotky si často nelogicky bránily v pohybu a tím způsobovaly neefektivní boj s nepřítelem. Například střelecké jednotky bránily v průchodu k nepříteli jednotkám, které mohly útočit pouze na blízké cíle. Takovýchto chyb v inteligenci se najde opravdu mnoho. Někdy se takovéto hloupé chování může vyskytnout i ve hrách, jejichž umělá inteligence je na vysoké úrovni. Je tedy těžké naprogramovat umělou inteligenci, která je bez jediné chyby. A jak již bylo řečeno, herní umělá inteligence musí být omezena na malý výpočetní výkon, tudíž algoritmy nemusí vracet optimální výsledek.

Na závěr této kapitoly chci napsat, že inteligence ve hrách je často na velmi dobré úrovni. Samozřejmě jsou na trhu hry, jejichž vojáci nevědí, že když nepřítel střílí, tak je lepší zůstat v zákrytu, ale pokud je úroveň zábavnosti hry vysoká, tak se tento nedostatek dá prominout.

3 Rozhodovací techniky

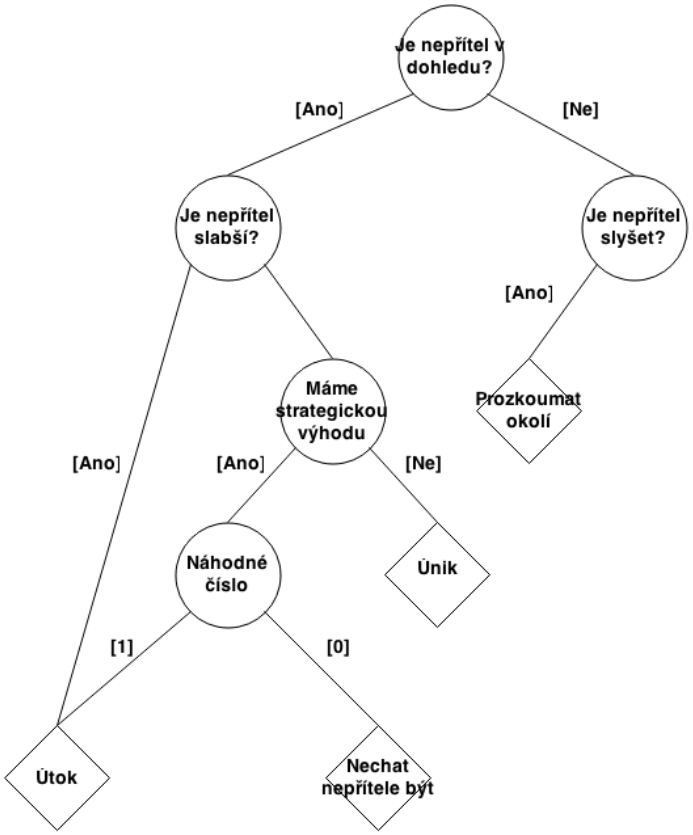
3.1 Rozhodovací strom

Rozhodovací strom (decision tree) je nejjednodušší z technik spojených s rozhodováním. Podle názvu je technika založena na stromové struktuře, kdy každý list stromu představuje akci, která se má vykonat. Rozhodování začíná v kořenu stromu a v každém uzlu, včetně kořene, se vyhodnotí podmínka, a podle toho, zda je splněna, se rozhodování přesune buď to do pravé, nebo levé větve. Z programátorského hlediska je nejvýhodnější použít tzv. binární strom. Tedy každý uzel má pouze dva následovníky. Možné je samozřejmě použít i jiný, než binární strom, ale všechny tyto stromy se dají jednoduše přepracovat na binární. Tato technika umožňuje jednoduše využívat náhodné složky. Řekněme, že pokud je splněna podmínka v některé větvi, tak se rozhodování může ubírat více směry. Například pokud nepřítel disponuje určitou technologií, a vhodných reakcí je více, jednoduše vygenerujeme náhodné číslo 0 nebo 1 a podle toho pokračujeme v rozhodování opět buď to v pravém, nebo levém podstromu. Tímto způsobem můžeme do tohoto jednoduchého systému vložit prvek překvapení. Reakce na stejné podněty tedy nebude vždy stejná, a tedy nemůže nepřítel jednoduše předpokládat reakci. Některé větve také nemusí vést k žádné akci. Také může být strom navrhnut tak, že do jednoto uzlu může vést více cest a tím zabránit duplicitním podstromům a zjednodušit tím celou strukturu. Obrázek 1 ukazuje jednoduchý rozhodovací strom.



Obrázek 1: rozhodovací strom

V obrázku 1 jsou kruhem značeny uzly a kosočtvercem listy (akce). Je patrné, že při záporné odpovědi na otázku, zda je nepřítel slyšet, strom neobsahuje žádnou akci. Obrázek 2 popisuje sofistikovanější strukturu využívající další možnosti této techniky.



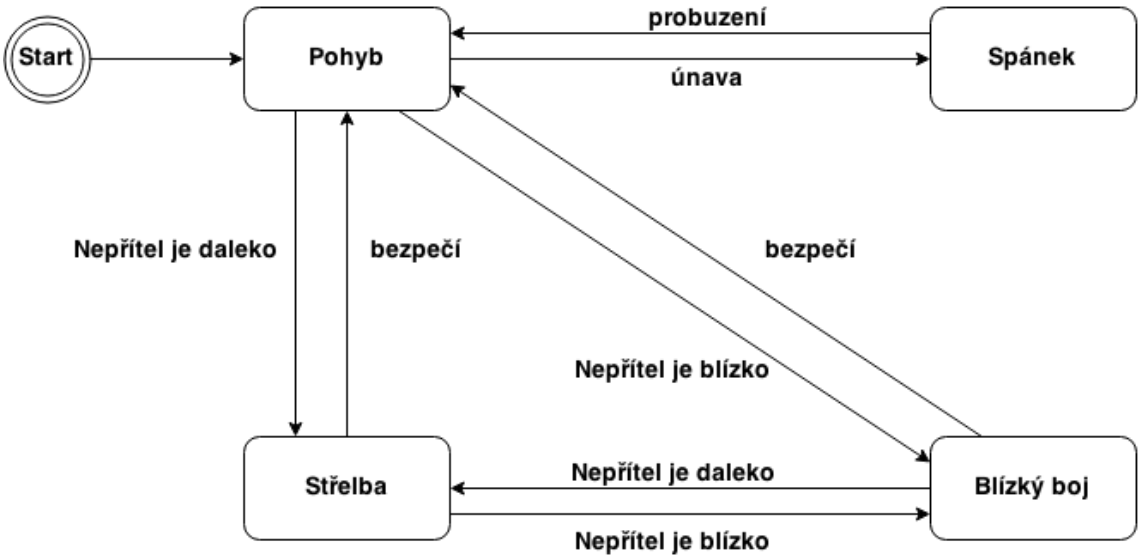
Obrázek 2: rozšířená struktura rozhodovacího stromu

Jak je vidět na obrázku 2, do akce „Útok“ se lze dostat dvěma cestami. Dále zde vidíme použitou náhodnou složku v uzlu, do kterého se dostaneme, když je splněna podmínka, že naše armáda disponuje strategickou výhodou.

Výhody této techniky jsou takové, že je velice jednoduchá na implementaci, je velice výkonná, snadno rozšiřitelná a velice přehledná.

3.2 Stavový automat

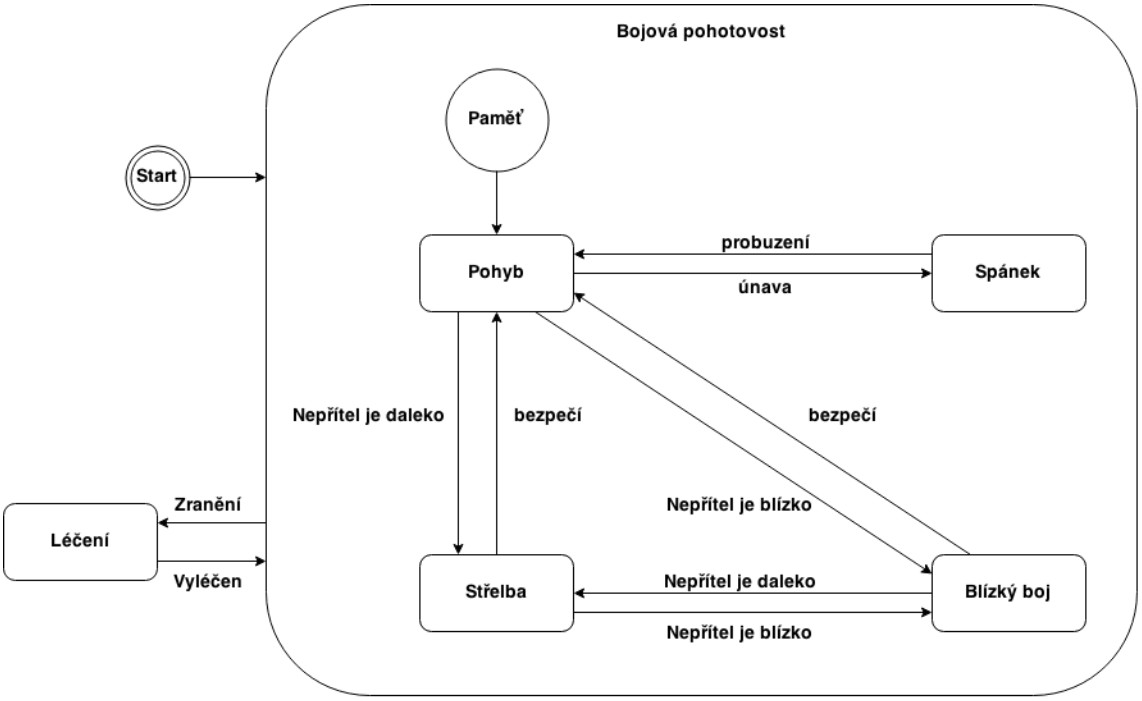
Stavový automat (state machine) je jednou z dalších rozhodovacích technik. Jak již název napovídá, skládá se ze stavů. Každý stav v sobě má akci, která je vykonávaná do té doby, dokud není splněna podmínka přechodu do stavu jiného. Stav třeba může spouštět určitý rozhodovací strom. Každý stav v sobě má množinu podmínek, které po splnění jedné zajistí změnu stavu automatu. Podmínky nazývejme přechodovými podmínkami, jelikož plní funkci právě přechodu z jednoho stavu do jiného. Obrázek 3 ukazuje strukturu základního stavového automatu.



Obrázek 3: stavový automat

Tento stavový automat ukazuje jednoduchý příklad bojovníka. Je zde vidět, že do některých stavů, jako je spánek, se automat může dostat pouze, pokud se nacházel ve stavu *Pohyb* a byla splněna podmínka únavy. Každý z těchto stavů obsahuje nějakou akci, která se neustále vykonává. Tedy pokud si bojovník drží nepřítele na delší vzdálenost, neustále střílí.

Existuje daleko více druhů takovýchto stavových automatů. Za zmínku stojí ještě „deterministický automat“. Představme si, že bojovník, jehož chování je reprezentováno stavovým automatem z obrázku 3 obdrží vážná zranění a jeho životy klesnou pod určitou úroveň, chceme, aby vypil léčivý lektvar. Každému z těchto stavů by bylo potřeba dodat přechod do stavu *Léčení*. Stavový automat by se tak ale stal nepřehledným.Deterministický automat zavádí tzv. úrovně, kdy jeden stav může v sobě obsahovat další stavy. Obrázek 4 znázorňuje tento přístup.



Obrázek 4: deterministický stavový automat

Obrázek 4 znázorňuje dvě úrovně. Ta nejspodnější byla znázorněna v obr 3. Vnější úroveň se skládá ze dvou stavů. Jeden je stav *Bojová pohotovost* a druhý je stav *Léčení*. Pokud z vnitřní úrovně zjistíme zranění, automat přejde do stavu *Léčení*. Po vyléčení životů se vracíme zpátky do stavu, ve kterém jsme se nacházeli naposled. Tento stav je uložen v tzv. Paměti. Ta slouží jako pamatování si posledního stavu, ve kterém se nacházel vnitřní automat před přechodem do stavu jiné úrovně. Slouží také jako úložiště pro stav spouštěný defaultně. Třeba při prvním vstupu do této úrovně stavového automatu. Dále se dá velice efektivně kombinovat stavový automat rozhodovacím stromem tak, že namísto některých akcí v listu stromu se objeví stav automatu. Jeto jedna s dalších účinných technik.

O technice stavového stromu lze říct, že je velice jednoduchá. [5] tuto metodu považuje za velice flexibilní, ale za cenu volání mnoha metod. Tyto metody mohou být polymorfní, což snižuje výkon ještě více. Na úsvitu herní umělé inteligence byl tento přístup hojně využíván a jedná se tedy o časem prověřenou techniku, která se využívá dodnes.

3.3 Cílově orientované chování

Do této doby jsem popisoval techniky, které příliš nebraly v potaz potřeby charakteru, kterého se inteligence týkala. Tento přístup cílově orientovaného chování (goal-oriented behavior) bere v potaz cíle charakteru, jeho potřeby. Tento přístup se velice hodí do her, jako simulátory života. Tento žánr představuje série The Sims [Maxis Software, Inc.]. V této hře není vytyčen žádný cíl. Hráč se stará o jednoho člověka, nebo skupinku lidí podobně, jako tomu bylo kdysi dávno, kdy bylo na výsluní dne Tamagoči. Tato technika zajišťuje uspokojení jakýchkoli potřeb charakteru. [5] popisuje, že charaktery potřebují jíst, když jsou hladové, spát, když jsou ospalé, psát si s přáteli, když jsou osamělé a potřebují obejmout, když potřebují lásku. Pokud bychom takovéto chování měli zajistit rozhodovacím stromem, tak by při velkém množství potřeb v proměnlivém prostředí byl zapotřebí rozsáhlý strom se stovkami parametrizovaných akcí. Účinnějším prostředkem je použití cílově orientovaného chování. Jelikož tento přístup není v praxi příliš používán, není obecně známá nejlepší implementace. Ale postup je následující:

- Musíme nadefinovat množinu potřeb, které může daný charakter uspokojovat
- Je zapotřebí mít nadefinovanou množinu akcí, kterými tyto potřeby uspokojíme, nebo pouze snížíme akutnost jejich potřeby uspokojit
- Každou potřebu, kterou je nutno v danou chvíli uspokojit je nutno ohodnotit nějakým číslem. Jedná se o jakousi sílu potřeby
- Musíme taky definovat, jak velké číslo potřeby je schopna akce uspokojit
- Hledáme množinu po sobě jdoucích akcí, které uspokojí co možná největší číslo potřeb

(Ian Millington, 2006, s. 387) znázorňuje tímto příkladem:

Potřeba: Uzdravení = 4
Potřeba: Zabít obra = 3
Akce: Ohnivá koule (Zabít obra: -2), zapotřebí 3 sloty magické energie
Akce: Malé léčení (Uzdravení: -2), zapotřebí 2 sloty magické energie
Akce: Velké léčení (Uzdravení: -4), zapotřebí 3 sloty magické energie
K dispozici: 5 slotů magické energie

Algoritmus by měl najít kombinaci „Malé léčení“, následované kouzlem „Ohnivá koule“. Tato kombinace využije přesně 5 slotů magické energie. Tuto kombinaci můžeme najít např. některou z heuristických optimalizačních technik, kterým tato práce dává prostor v pozdějších kapitolách.

Opět existuje více typů této techniky. Může se do hledání zapojit v potaz časová složka. První akce nám zabere 10 časových jednotek, za kterých se může doposud nižší

nutnost potřeby stát kritickou. Dále z důvodu nedosažitelnosti plného uspokojení můžeme omezit počet akcí v hledané sekvenci, atd.

Tento nepříliš používaný postup může zajišťovat chování, které by bylo ostatními technikami velice obtížně dosaženo. Jeho nevýhodou může ale být výpočetní složitost hledání posloupnosti akcí. Avšak za použití vhodných heuristik se výkon rapidně zvýší a může tím cílově orientované chování nabýt na použitelnosti.

3.4 Systém založený na pravidlech

Systém založený na pravidlech (Rule-Based System) je technika známá už od 70 let 20 století. Navzdory její neefektivitě a složitosti naprogramovat, byly tyto systémy hojně využívané. Jedním odvětvím tohoto systému jsou také tzv. *expertní systémy*. V 80. letech 20. století byly expertní systémy téměř synonymem k umělé inteligenci popisuje [8]. Technika systémů založených na pravidlech je postavena na databázi znalostí, kterými bot disponuje, a sadou pravidel, mezi kterými se hledá shoda s databází. Někdy se taky využívá tzv. *arbitr*, který určuje, která reakce na shodu pravidla s databází znalostí bude spuštěna. Databáze znalostí:

Legolas má 100 životů
Aragorn má 150 životů
Gimli má 200 životů
Frodo má 15 životů

Databáze zobrazuje stav skupiny. Frodo je pověřen důležitým úkolem nést prsten. Taky je ale na pokraji života. Našim cílem je v případě ztráty Froda předat prsten Aragornovi. Abychom docílili takovéhoto chování, musíme do databáze přidat záznam o držení prstenu.

Frodo nese prsten.

V naší sadě pravidel budeme mít pravidlo:

IF Frodo má životů < 5 AND Frodo nese prsten THEN Akce: Frodo předej prsten Aragornovi

Jak je tedy vidět, naše sada pravidel se řídí příkazy IF (pokud) THEN (potom). Využívá logických operací jako AND (a zároveň), OR (nebo), NOT (zápor) a podobně. Po akci je třeba upravit aktuálnost databáze. Tzn. smazat záznam o Frodem drženém prstenu a na místo něj uložit záznam „Aragorn nese prsten“.

Pro zobecnění se v takovýchto systémech používají *wildcards* (divoké karty), které jsou použity třeba v případě, když potřebujeme mít pravidlo

Kdokoli má životů < 10

Jak již bylo naznačeno, často se v takovýchto systémech vyskytují arbitři . Tím nejjednodušším je ten, který spustí akci při první nalezené shodě databáze znalostí s pravidlem. Jinou strategií může být spouštění akce nejméně krát shodujícího se pravidla, nebo spouštění akce náhodné shody z nalezených a podobně. Je tedy celá řada možností, jak tento systém modifikovat.

Tento systém je velice náročný jak na implementaci, tak na výkon. Navzdory této skutečnosti to byl velice oblíbený systém. Dá se ovšem velice jednoduše nahradit rozhodovacím strojem, nebo stavovým automatem.

3.5 Závěr kapitoly

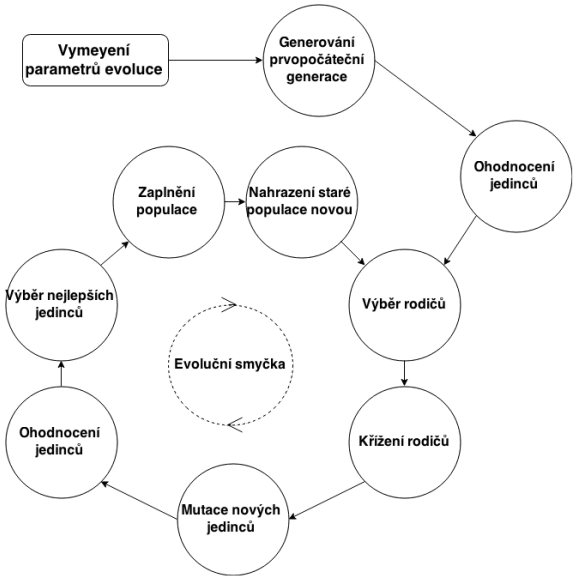
Z předchozích řádků je patrné, že existuje mnoho technik spojených s rozhodováním. Mnohé z nich jsem z nich ani nezmínil. Mezi další techniky patří třeba *black board systém*. Ten obsahuje mnoho specialistů na různé věci. Třeba specialista na pohyb, specialista na střelbu. Ti se musí dohodnout, kdo bude provádět další činnost podle vhodnosti situace. Nezmínil jsem ani *fuzzy logiku*, což je nadmnožina klasické Booleanovské logiky. Je to spíše matematika používaná k rozhodování. Dále se používají *neuronové sítě*, které napodobují proces, který probíhá v mozku při procesu myšlení. Dále třeba skriptování. [3] popisuje jeho cíl, jako umožnění uživatelům ovládat a přizpůsobit chování aplikace pomocí skriptovacího jazyka. Jak je vidět, tak seznam nekončí. A pro vývoj dobré herní umělé inteligence je potřeba také vybrat vhodný prostředek k její tvorbě.

4 Evoluční výpočetní techniky

Jako další možností při tvorbě umělé inteligence může být použití evolučních výpočetních technik, dále jen EVT. „Evoluční výpočetní techniky jsou numerické algoritmy, které vycházejí ze základních principů Darwinovy a Mendelovy teorie evoluce, jejíž hlavní ideou je předávání rodičovského genomu novým potomkům a následné uvolnění životního prostoru potomkům.“ [10] Tyto techniky jsou postaveny na tzv. evolučních algoritmech (evolutionary algorithms, EA), které v podstatě tvoří většinu EVT. Tyto techniky koexistují spolu s rozšířeními, jako evoluční hardware, genetické programování atd. EA jsou prostředkem pro řešení problémů, které by byly řešitelné konvenčními algoritmy jen obtížně, nebo těmito prostředky zcela neřešitelné. Třeba z důvodu výpočetní složitosti. Tyto algoritmy většinou spadají do sorty stochastických algoritmů, je v nich tedy zahrnuta nějaká náhodná složka.

4.1 Obecný princip

Podle klasické Darwinovy a Mendelovy teorie evoluce je uznáváno evoluční dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. „Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají „cyklicky“ po tzv. generacích (generations), čímž uvolňují místo novým lepším potomkům, kteří se stávají novými rodiči.“ [10] Evoluční principy jsou znázorněny na Obr. 5. Na obrázku není naznačeno ukončení evoluce po n generacích a výběr nejlepšího jedince – řešení z poslední populace



Obrázek 5: Obecný cyklus evolučního algoritmu.

Použití evolučních principů pro účely složitých výpočtů má následující postup (podle Obr. 5):

1. Vymezení parametrů evoluce: v tomto kroku jsou stanoveny parametry, které řídí běh algoritmu. Jsou to také ukončovací podmínky např. počet generací, minimální odchylka nejlepšího a nejhoršího jedince populace. Dále se zde definuje účelová funkce (cost function), případně tzv. vhodnosti (fitness – upravená návratová hodnota účelové funkce). Účelovou funkcí je myšlen obvykle matematický model problému, jehož minimalizací, či maximalizací (obecně tedy extremalizace) dostaneme řešení problému. „Tato funkce s případnými omezujícími podmínkami je jakýmsi „ekvivalentem životního prostředí", v němž se vyhodnocuje kvalita aktuálních jedinců." [10] Nebo [4] ji popisuje jako zvláštní druh objektivní funkce, která kvantifikuje optimálnost řešení.

2. Generování prvopočáteční populace: populací je obecně míněna matice MxN, kde N je počet parametrů jedince. Tento parametr se bude dále označovat D. Budeme se tedy spíše bavit o dimenzi účelové funkce. Písmeno M značí počet jedinců v populaci. Podle počtu argumentů optimalizované funkce a dalších uživatelských kritérií je vygenerována počáteční populace jedinců (individuals). Jedincem se myslí vektor čísel, který má tolik složek, jako počet optimalizovaných parametrů účelové funkce. Parametry jedince jsou vygenerovány jako náhodné hodnoty, a celý jedinec tedy reprezentuje jedno konečné řešení problému. Tito jedinci tedy tvoří populaci (population).

3. Ohodnocení jedinců: „Všichni jedinci se ohodnotí přes definovanou účelovou funkci a každému z nich se přiřadí a) buď přímá hodnota vrácená účelovou funkcí, nebo b) vhodnost, což je upravená (obvykle normalizovaná) hodnota účelové funkce." [10]

4. Výběr rodičů: Tento výběr je proveden na základě jejich kvality (vhodnost, hodnota účelové funkce), popř. i dalších kritérií.

5. Křížení rodičů: Křížení dává vzniku novým jedincům. Tento proces je odlišný u každého EA. U klasických genetických algoritmů jsou přehozeny části rodičů, u diferenciální evoluce je křížení jistou vektorovou operací apod. Křížení v klasických EA je znázorněno v Obr. 6

6. Mutace nových jedinců: V tomto bodě jsou všichni noví jedinci podrobeni mutaci. Vhodně zvolený náhodný proces pozmění jedince. [10] poukazuje na ekvivalent tohoto procesu s biologickou mutací genů jedince. Tento děj popisuje obr. 7

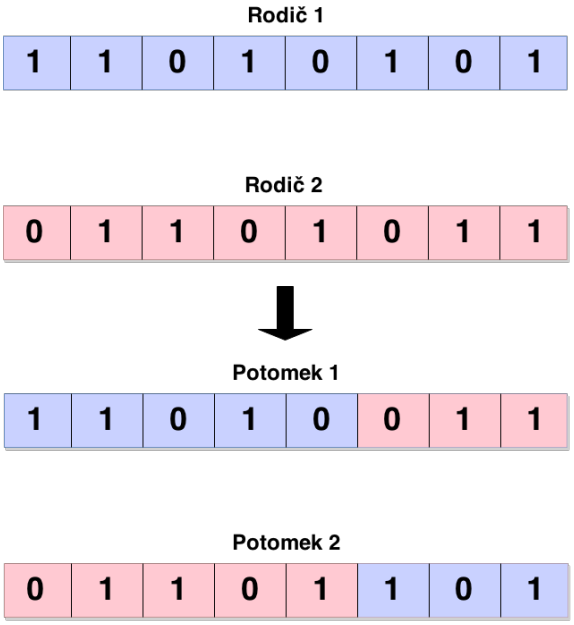
7. Ohodnocení jedinců: Každý jedinec je ohodnocen stejně, jako v kroku 3.

8. Výběr nejlepších jedinců: Vyberou se nejlepší jedinci (leadeři) populace.

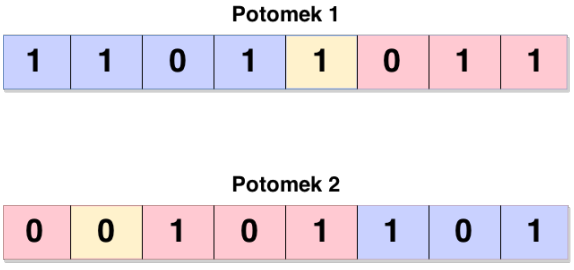
9. Zaplnění nové populace: Vybraní jedinci zaplní novou populaci.

10. Nahrazení staré populace novou: Stará populace je nahrazena novou a pokračuje se opět krokem 4.

Kroky 4 – 10 se opakují tak dlouho, dokud není dosažena potřebná kvalita řešení, nebo je dosažen uživatelem zadaný maximální počet generací.



Obrázek 6: Křížení v klasických genetických algoritmů



Obrázek 7: Příklad mutace

Obrázek 6 ukazuje příklad křížení, kdy si potomci pouze vymění náhodně velkou část rodičů a obr. 7 ukazuje jejich následnou mutaci tak, že se každému potomkovi zvolí

náhodná pozice přehození hodnoty. Křížení a mutace se různí podle konkrétního evolučního algoritmu. Mutace u algoritmu SOMA se je třeba prováděná rušením migrace jedince putujícím po účelové funkci. U diferenciální evoluce je třeba křížení prováděno pomocí pří rodičů. Existují i výjimky, které se neřídí body 1 – 10, v takovém případě se o příslušných algoritmech nemluví jako o algoritmech evolučních, ale jako o algoritmech, které patří k EVT. Obvykle. „Někteří evolucionisté je vyčleňují z třídy EVT úplně. Příkladem může být algoritmus ACO (Ant Colony Optimization) – optimalizace mravenčí kolonií, který napodobuje chování mraveniště a umí řešit extrémně složité kombinatorické problémy. Je založen na principech spolupráce více jedinců patřících ke stejnému společenství. V tomto případě mravenců.“ [10]

Evoluční algoritmy nenabyly na oblíbenosti jen z důvodu jejich modernosti a nekonvenčnosti. Oblíbenost si nabyly hlavně pro fakt, že v případě vhodného aplikování mohou nahradit člověka. Jeden z evolučních algoritmů jsem použil ve své práci vytvoření inteligence pro dříve zmiňovanou hru StarCraft: Brood War. Je to algoritmus SOMA.

4.2 SOMA – Self organising migration algorithm

SOMA, samo organizující se migrační organismus je, podobně jako třeba *diferenciální evoluce* nebo *rojení částic* (další algoritmus patřící do EVT), založena na vektorových operacích. „Vzhledem k tomu, že pracuje s populacemi podobně jako např. genetické algoritmy a výsledek po jednom evolučním cyklu (v tomto případě nazvaným migrační kolo) je totožný s genetickými algoritmy či diferenciální evolucí (je vytvořena nová populace), lze jej řadit např. mezi evoluční algoritmy navzdory faktu, že během jeho běhu nejsou z hlediska filozofie algoritmu vytvářeni noví potomci, jak je tomu u jiných klasických evolučních algoritmů.“ [10] SOMA se řadí mezi algoritmy memetické[3] , nebo hejnové (swarm inteligence). Algoritmus SOMA je postaven na principech pozorovaných v přírodě, kterými se v sociálně-biologickém prostředí řídí inteligentní jedinci, kteří se kooperativně podílí na společném úkolu. SOMA se od ostatních EA liší v tom, že nová populace není tvořena nástrojem evoluce křížením. Jde spíše o kooperativní prohledávání (migraci) v prostoru možných řešení daného problému.

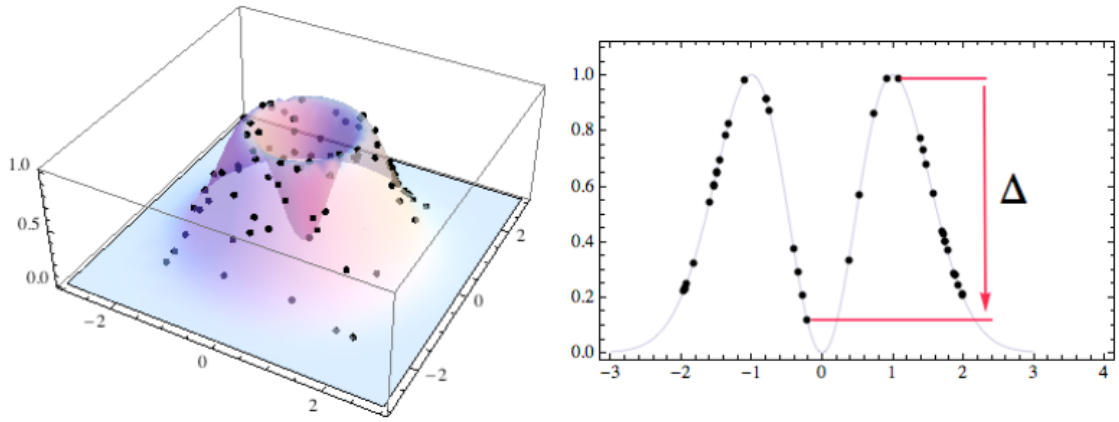
4.2.1 Parametry algoritmu SOMA

Stejně jako u ostatních evolučních algoritmů, je běh ovlivňován speciální množinou parametrů, které se dají rozdělit do dvou skupin. Řídicí parametry a ukončovací parametry. Řídící parametry silně ovlivňují kvalitu běhu algoritmu a ukončovací parametry jsou ty, které běh algoritmu ukončují. Všechny tyto parametry musí být zvoleny uživatelem ještě před započetím běhu algoritmu. Jejich přehled je znázorněn tabulkou 1.

[3] Memetika je disciplínou, usilující o evoluční modelování přenosu informací v kultuře

Parametr	Doporučený rozsah	Poznámka
PathLength	$\langle 1, 1; 5 \rangle$	Řídicí parametr
Step	$\langle 0, 11; PathLength \rangle$	Řídicí parametr
PRT	$\langle 0; 1 \rangle$	Řídicí parametr
D	dáno problémem	Počet argumentů účelové funkce
PopSize	$\langle 10; definujeuivatel \rangle$	Řídicí parametr
Migrace	$\langle 10; definujeuivatel \rangle$	Ukončovací parametr
MinDiv	definuje uživatel	Ukončovací parametr

Tabulka 1: Význam parametrů SOMA Zdroj: [10].



Obrázek 8: Princip ukončovacího parametru *MinDiv*, Zdroj: [10]

Černé body na obrázku 8 jsou jedinci populace a jestliže platí $\Delta < MinDiv$, pak se ukončí běh algoritmu SOMA.

PathLength (délka cesty) Parametr popisuje, v jaké vzdálenosti se aktivní jedinec na své cestě zastaví od tzv. vedoucího jedince. Při $PathLength = 1$ se jedinec zastaví na pozici vedoucího jedince, při $PathLength = 2$ se jedinec zastaví za vedoucím jedincem ve vzdálenosti, z jaké jedinec startoval. Pokud bude *PathLength* nastaven na hodnotu menší, než jedna, jedinec se zastaví před vedoucím jedincem a povede to k degradaci migračního procesu. V takovém to případě algoritmus bude nalézat pouze lokální extrémy a ne ty globální. Podle [10] velikost $PathLength = 3$ byla zjištěna jako dostačující prakticky ve všech simulacích na všech problémech.

Step (krok), Parametr step určuje granualitu, se kterou budou jedinci populace prohledávat účelovou funkci při jejich migraci. Při nastavení parametru step na menší

hodnotu budou jedinci svou cestu mapovat podrobněji, naopak při nastavení parametru step na velkou hodnotu, budou jedinci svou cestu mapovat jen velmi hrubě. „Je rovněž důležité nastavit Step tak, aby vzdálenost mezi vedoucím a aktivním jedincem nebyla celočíselným násobkem parametru Step. Pokud by se tak stalo, diverzibilita populace by klesla, protože každý jedinec by mohl být finálně přitažen vedoucím jedincem a proces by tak mohl rychleji skončit v lokálním extrému.“ [10] Proto je preferováno nastavení hodnoty *step* na $0,11$, než na $0,1$.

PRT (pertrubace): Dle tohoto parametru se tvoří tzv. pertrubační vektor (PRTVector), který slouží jako rušící vektor při pohybu jedince a ovlivňuje, zda bude pohyb jedince směřovat přímo k vedoucímu jedinci, či ne. Tento parametr patří mezi nejdůležitější parametry tohoto evolučního algoritmu. Jeho optimální hodnota je kolem hodnoty 0,1. Čím více se *PRT* blíží hodnotě 1, tím více narůstá konvergence jedinců k vedoucímu jedinci a algoritmus poté směřuje pouze k lokálnímu extrému.

D je parametr, který udává počet optimalizovaných parametrů, tedy parametrů účelové funkce. Je dán problémem samotným a změněn může být pouze tehdy, když předefinujeme celý problém.

PopSize (velikost populace), parametr udává počet jedinců v populaci. Obecně může být tento parametr nastaven na hodnou 2, v takovémto případě se ale algoritmus chová jako klasický deterministický algoritmus. Obvykle by parametr neměl klesnout pod hodnotu 10 z důvodu výkonnosti algoritmu samotného.

Migrace ten to parametr patří mezi ukončovací a jednoduše definuje, kolik migračních kol algoritmus maximálně provede. Ostatní EA obsahují tento parametr pod názvem „generace“, ale z hlediska chování algoritmu SOMA byl tento název zvolen pro jeho výstižnost.

MinDiv (minimální diverzita), tento ukončovací parametr určuje, jaký maximální rozdíl mezi ohodnocením nejlepšího jedince a nejhoršího jedince je povolen. Pokud je tedy rozdíl menší, než je hodnota tohoto parametru, algoritmus končí. Tento parametr se většinou nastavuje na male hodnoty. Pokud by byl nastaven nesprávně na hodnotu příliš velkou, mohlo by se stát, že by algoritmus skončil po prvním migračním kole a vracel by tedy nesmyslný výsledek.

4.2.2 Mutace

Mutace je nezbytnou součástí všech EA, avšak u každého z nich probíhá jiným způsobem. Společný rys ale nalezneme v tom, že právě zde se nachází stochastická složka. Je zde použit generátor náhodných čísel k tomu, aby náhodně změnil jedince. U algoritmu SOMA je tento pojem přejmenován na pertrubaci (rušení). Pohyb jedinců na hyperploše[4] je náhodně rušen – pertrubován, nikoli mutován. Výsledek je stejný, jde spíše o terminologii. Tabulka 2 popisuje funkci parametru *PRT*, který udává sílu pertrubace.

rnd	PRTVector
0,207	1
0,587	0
0,856	0
0,042	1

Tabulka 2: Příklad perturbačního vektoru pro jedince o 4 parametrech a $PRT = 0,3$

Je-li náhodně vygenerovaná hodnota menší, jako hodnota parametru *PRT*, bude na aktuální pozici *PRTVektoru* zapsána hodnota 1, pokud bude náhodná hodnota větší, než je hodnota *PRT*, bude zapsána 0. Efekt *PRT* vektoru je ten, že obvykle eliminuje pohyb jedince z *N* rozměrné plochy na *N – k* rozměrnou plochu, což paradoxně zvyšuje šanci nalezení globálního extrému. *PRTVektor* je modifikován před každým skokem jedince.

4.2.3 Křížení

„U klasických evolučních algoritmů znamená pojem „křížení“ tvorbu nového potomka neboli vygenerování nové pozice na dané hyperploše pomocí dvou již existujících pozic (rodičů).“ [10] Jde tedy o vygenerování nového řešení, které může být lepší, nebo naopak horší, než kvalita řešení „rodičů“. To poté vede k potvrzení, nebo zamítnutí v nové populaci. SOMA je zde odlišná, než ostatní EA. Jakási sekvence potomků je vygenerovaná na cestě jedince. Je-li $PathLength = 14$ a $Step = 0,11$, je tedy vygenerováno 10 potomků, ze kterých je do další populace vybrán ten nejlepší, podle jejich vhodnosti. Může se se stát, že jako potomek postupující do nové generace je onen původní jedinec. Tedy jedinec s parametry, které obsahoval před započetím jeho cesty. Jak tedy bylo naznačeno, filozofie křížení je zde nahrazeno generováním sekvence potomků. Při tomto pohybu si jedinci pamatují nejlepší nalezenou pozici a tato se poté stává pozicí jedince v dalším migračním kole, jak tvrdí [10]. Jedincova cesta je definována rovnicí 1. Jedná se o rovnice směrového vektor.

[4] prostor možných řešení

$$x_{i,j}^{ML+1} = x_{i,j,start}^{ML} + (x_{L,j}^{ML} - x_{i,j,start}^{ML+1})tPRTVector \quad (1)$$

$x_{i,j}^{ML+1}$ následující pozice jedince na cestě

$x_{i,j,start}^{ML}$ startovní pozice jedince (pozice před započetím jeho cesty hyperplochou)

$x_{L,j}^{ML}$ pozice leadera (vedoucího jedince) Počet hodnot, kterých může t nabývat je $\frac{PathLength}{Step}$.

$t \in \langle 0; PathLength \rangle$ t se zvětšuje po hodnotě Step

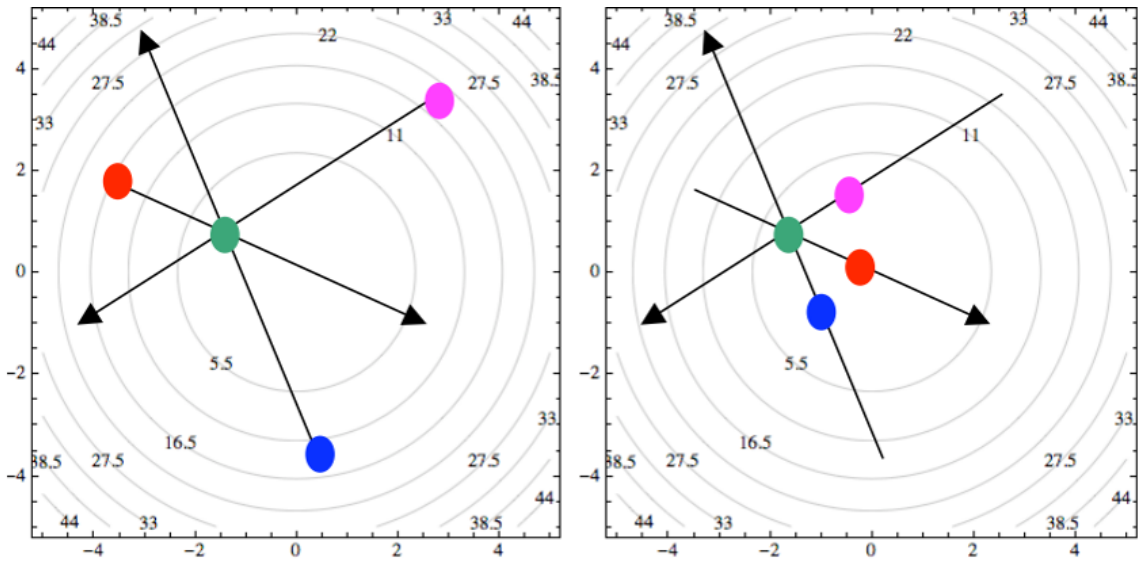
4.3 Princip algoritmu SOMA

Jak již dříve bylo naznačeno, algoritmus SOMA byl inspirován soutěživě-kooperativním chováním inteligentních jedinců řešící stejný problém. [10] uvádí příklad tohoto chování u smečky lovících vlků, včelího úlu, kolonií termitů apod. V těchto případech je společný cíl např. nalezení nejlepší potravy. Lovící smečka spolupracuje za účelem úkol splnit. Jedinci smečky si navzájem sdělují kvalitu jejich potravy (řešení) a přitom nevědomě soutěží o to, kdo bude tím, který nalezne ono nejlepší řešení. Znalost aktuálně nejlepšího zdroje potravy ovlivňuje chování jedince při putování na vlastní cestě. Jedinec s nejlepším ohodnocením „přitahuje“ ostatní jedince putující vlastní cestou. Jinými slovy jedinci na své cestě konvergují k leaderovi. Během této své cesty (migrace) se snaží najít ještě lepší zdroj potravy. Tabulka 3 porovnává terminologii v algoritmu SOMA s biologickou terminologií.

Biologická realita	**Počítačová implementace**
členové smečky, společenství	jedinci v populaci, parametr *PopSize*
člen společenství s nejlepším zdrojem potravy	Leader, vedoucí aktuálního migračního kola
potrava	vhodnost, hodnota účelové (kriteriální, cenové) funkce, geometricky je to lokální či globální extrém na N rozměrné hyperploše
životní prostor společenství	Hyperplocha daná účelovou funkcí
migrace členů společenství v životním prostředí	Migrační kola v algoritmu SOMA

Tabulka 3: Význam biologické terminologie v algoritmu SOMA, Zdroj: [10]

SOMA pracuje v cyklech, nazývajících se migrační kola. Jejich maximální počet je specifikován zmíněným parametrem *migrace*. Je to významový ekvivalent generací.



Obrázek 9: Princip SOMA, Zdroj: [10]

Obr. 9 znázorňuje umělý příklad, vlevo před migrací, vpravo po migraci na nových pozicích.Zelenou barvou je zde označen leader, ostatními barvami jsou označeni jedinci populace na své cestě. Šipky poté značí směrový vektor k leaderovi. Ten je pertrubován PRTVektorem. Příklad na obrázku značí případ, kdy tento vektor je jednotkový, tedy omezuje pohyb jedince z N parametry na N-k parametrů, kde k=0. Obrázek popisuje strategii *AllToOne* (všichni k jednomu). Tato strategie je základní strategií SOMA. Vedle ní však existují další strategie jako *AllToAll* (všichni ke všem), *AllToAllAdaptive* (všichni ke všem adaptivní), *AllToOneRand* (všichni k jednomu náhodně), *Clusters* (svazky) a další. Základní verze algoritmu se skládá z následujících kroků:

1. **Definice parametrů:** zvolení řídicích a ukončovacích parametrů zmíněných dříve
2. **Tvorba populace:** V tomto kroku je vytvořena prvopočáteční populace pomocí generátoru náhodných čísel. Za pomoci Specimenu a generátoru je pro každý parametr jedince generováno náhodné číslo, jehož typ a hodnota je dána rozsahem v odpovídajícím prvku Specimenu.
3. **Migrační kola:** Jedinci jsou ohodnoceni vhodností a ten nejlepší je zvolen za leadera pro následující migrační kolo. V tomto momentu se jedinci populace začnou pohybovat směrem k leaderovi pomocí skoků, jejichž velikost určuje parametr Step. S každým skokem si jedinec spočítá novou hodnotu účelové funkce. Jedinec skočí tolikrát, dokud není dosaženo pozice, která byla dána parametrem *PathLength*. S každým jeho skokem je vygenerován *PRTVektor*, který ruší pohyb směrem k leaderovi. Na konci své cesty se jedinec vrací na místo, kde byl ohodnocen nejlépe. Toto místo je poté výchozím bodem do nového migračního kola.

4. **Testování naplnění ukončovacích parametrů:** Testuje rozdíl vhodností mezi nejlepším a nejhorším jedincem. Pokud je tento rozdíl menší než hodnota parametru *MinDiv*, algoritmus končí stejně tak, jako končí v případě dosažení počtu migrací daných parametrem *Migrace*.

5. **Stop:** Po ukončení migračních kol je za výsledek považován zvolený leader po posledním migračním kole.

5 Seznámení s hrou

Hra Starcraft je RealTimovou strategií (RTS). To znamená, že hráč ovládá své jednotky v aktuálním čase. Tedy nepřetržitě. Existuje ještě jeden druh strategických her, kterým je „Tahová strategie“. Ta se zase vyznačuje tím, že hráč své jednotky ovládá v tazích a takováto hra je často hrána ve více lidech na jednom počítači. Tito hráči se v tazích střídají. Hra více hráčů na jednom počítači je v případě real time strategie možná pouze v případě, že by se obrazovka rozdělila na poloviny a každému hráči by příslušela jedna polovina. Ale z charakteristiky takovýchto her je to nevýhodné. Totiž jedná se o strategickou hru. Druhý hráč by neměl vědět o chování nepřítele, jeho pozicích, technologiích, kterými disponuje a podobně. Hra se řadí mezi *military science fiction*. Jak už anglický název napovídá, jedná se o válečnou hru s prvky science fiction.

Hru vydala dnes světoznámá herní společnost Blizzard Entertainment 31. 3. 1998 na Windows a o rok později vyšla i na Mac OS. Společnost Blizzard vydala již mnoho známých titulů, jako třeba třídílnou sérii Diablo (RPG), nejúspěšnější MMORPG v historii World of Warcraft a také druhý díl hry StarCraft, který je koncipovaný do tří příběhových linií, dle ras. První díl StarCraftu se zařadil mezi nejprodávanější hru na osobní počítač v historii.

Příběh se odehrává na přelomu 25. a 26. století kde hlavní dějová linie se odvíjí kolem konfliktu tří velkých. Těmi civilizacemi jsou *Terrani*, lidé vyhoštění ze Země, *Zergové*, obří členovci s cílem dosáhnout genetické dokonalosti, a *Protosové*, humanoidní vysoce pokročilá civilizace s psionickými schopnostmi.

5.1 Popis hry

Hra se soustřed'uje na získání kontroly nad zdroji, které jsou nutné k budování bojových jednotek a staveb k nim potřebným. Existují zde dva druhy zdrojů. Nerosty neboli *minerály*, a plyn *vespene*. Minerály jsou potřeba skoro na všecky jednotky, budovy vylepšení a plyn až na pokročilé jednotky a vylepšení. Minerály jsou modré krystaly rostoucí ze země, které mohou těžit dělníci. Na to, aby mohli dělníci těžit plyn, je zapotřebí ještě patřičná budova. Plyn se vyskytuje v podobě gejzírů se zelenými oblaky.

Každá jednotka má určité požadavky na zásobování, jež souvisejí se sílou jednotky a hráči musí udržovat dost těchto podpůrných jednotek. Tyto podpůrné jednotky zajišťují určité budovy. Výjimku tvoří Zergové, kteří mají zásobování zajištěno speciální jednotkou, která je takovou létající stavbou pro zásobování a podporu. Umí například odhalit neviditelné, nebo zahrabané jednotky. Obrázky 10, 11, 12 ukazují tyto zásobovací budovy. Hráči budují infrastrukturu základny. Některé budovy slouží jako ochrana před nepřítelem ať pozemním, nebo vzdušným, některé jsou klíčem k vytváření pokročilých jednotek, jiné dávají možnost vylepšení některých schopností jednotek, nebo budov. Zpřístupňují kouzla, nebo zlepšují atributy jako obrana, poškození, dostřel, rychlost pohybu, rychlost útoku apod. Jiné budovy, jak už bylo řečeno, zajišťují zásoby, které jsou nezbytné pro stavbu jednotek. Dále pak hlavní budova, která dokáže vytvářet dělníky. Také slouží jako místo, kde dělníci svážejí natěžené suroviny.

Každá rasa je něčím unikátní. Protosové stavějí silné, výdržné, ale drahé jednotky, zatímco Zergové spoléhají na totální přečíslení nepřítele a rychlost. Terani jsou flexibilní alternativou k oběma zmíněným rasám. Jsou kompromisem mezi množstvím a kvalitou.

Obrázek 10: Protoská zásobovací budova: Pylon

Obrázek 11: Terranská zásobovací budova: Supply Depot

Obrázek 12: Zergská zásobovací jednotka: Overlord

5.1.1 Protosové

Jak bylo už zmíněno, Protosové produkují velice silné jednotky, zato ale velice drahé. Vedle svých životů disponují ještě štíty, které se mohou regenerovat. Svoje budovy musí mít v dosahu některého z Pylonů na obr. 10, což je pro Protosy také budova zajišťující zásoby. Výhodou Protosů může být fakt, že jejich budova zajišťující obranu je jak proti pozemním jednotkám, tak proti vzdušným jednotkám. Také dokáže odhalit neviditelné, nebo zahrabané jednotky. 13 ukazuje hlavní budovu a dělnické jednotky. Ukázka protoské základny v pokročilém stádiu je v příloze A.1.

Obrázek 13: Protoská hlavní budova dělnická jednotka: Probe

5.1.2 Terani

Výhodou Teranů může být jejich flexibilita. Produkují jednotky vyvážené a převážně střelecké. Tato rasa má velice silnou obranu zaručenou jednotkou v podobě tanku, který se může přetransformovat v dalekonosnou artilérii. Jejich budovy mohou, až na některé, létat a přemisťovat se tak na mapě. Obr. 14 ukazuje hlavní budovu a dělnické jednotky. Ukázka terranské základny v pokročilém stádiu je v příloze A.2.

Obrázek 14: Terranská hlavní budova dělnická jednotka: SCV

5.1.3 Zergové

Zergové jsou rasou, která vítězí počtem a rychlostí. Jejich jednotky jsou převážně slabé a levné. Produkují až na pár výjimek jen pozemní jednotky, což může být nevýhodou. Tyto jednotky se ale po vylepšení v hlavní budově mohou zahrabat. Tento efekt je podobný neviditelnosti s rozdílem, že takto zahrabané jednotky se nemohou hýbat. Zergové mohou své budovy stavět výhradně na slizu, který produkují některé jejich budovy. Obr. 15 ukazuje hlavní budovu a dělnické jednotky. Všechny zergské jednotky se líhnou z larev ukázaných taktéž na obr. 15. Ukázka zergské základny v pokročilém stádiu je v příloze A.3.

Obrázek 15: Zergská hlavní budova dělnická jednotka: Drone

Další obrázky ze hry jsou v příloze A.4. Více informací o hře na `http://classic.battle.net/scc/`

6 Implementace

Mým cílem bylo navrhnout umělou inteligenci do strategické hry StarCraft, použít pro to techniky spjaty s problematikou umělé inteligence a otestovat jejich vhodnost na konkrétním problému. Jinými slovy jsem měl navrhnout hráče ovládaného počítačem, chovajícího se jako člověk. Tomuto hráči říkejme bot. Tento bot by měl být schopný si zajistit suroviny, armádu, řídit se nějakou z ověřených strategií, efektivně bojovat, prohledávat mapu, zjišťovat informace o nepříteli, případně adekvátně reagovat. Tyto věci nám mohou při správném použití zajistit techniky popsány dříve. Jimi jsou tedy např. evoluční výpočetní techniky, rozhodovací stromy, stavové automaty, fuzzy logika, mechanizmy cílově orientovaného chování, na pravidlech založené systémy, blackboard systémy, skriptování atd.

Bot by se měl skládat z tří hlavních komponent. Pohybové komponenty, komponenty pro rozhodování a strategie, přičemž jedna komponenta může pokrývat více z těchto oblastí. Pro rozhodování jsem použil rozhodovací strom (decision making tree) z důvodu jeho jednoduchosti, strategie je zaručena jak pomocí předdefinovaných sekvencí akcí, které slouží pro počáteční fázi hry. Pohyb zajišťuje evoluční algoritmus jménem SOMA ve specifické implementaci. Propojení vlastního kódu ze hrou zajišťovalo rozhraní JNIBWAPI ,psáno v Javě, využívající nativních knihoven psaných v jazyce C++. Rozhraní JNIBWAPI je nadstavbou na rozhraní BWAPI, které je psáno právě v jazyce C++, a poskytuje jen další možnost jazyka pro psaní bota do hry StarCraft: Brood War. Více informací najdete na odkazu `https://code.google.com/p/bwapi/`

Každá z rozhodovacích technik produkuje tzv. *akce*, což je třída popisující činnost, nebo sadu činností, které se mají vykonat ať sériově za sebou, nebo paralelně. Každá vyprodukovaná akce je zpracována exekučním managerem, který centralizuje zpracovávání akcí. Celek tak udává jednoduchou, přehlednou strukturu, která je jednoduše rozšiřitelná.

Rozhraní BWAPI obsahuje metodu „act()“, která se periodicky opakuje vždy po uplynutí nějakého stálého časového okamžiku. Řekněme po jedné sekundě. To dává impulz rozhodovacímu stromu, aby rozhodl, jaká akce se má provést. Dále pak obsahuje metodu „gameUpdate()“, která se volá s každým snímkem a je vhodná pro ladící výstupy přímo do hry. Třeba vykreslení modrého kruhu kolem dělníka těžícího minerály, červeného kruhu značící jedince evolučního algoritmu apod.

Implementace bota včetně videí ze simulací se nachází v příloze B.

6.1 Pohyb

Pohyb je velmi důležitou součástí umělé inteligence a v tomto případě byl řešen evolučními výpočetními technikami. Z důvodu chování jednotlivých EA bylo možno použít některý ze dvou algoritmů. Rojení částic nebo SOMA. Vyplývá to z toho důvodu, že oba tyto algoritmy simulují pohyb jedinců po jakýmsi „životním prostředím“ daným účelovou funkcí. Rojení částic simuluje pohyb hejna ptáku, a SOMA zase reprezentuje vzorec chování lovící smečky. Hra StarCraft je spíše bojovou hrou, proto byl použit algoritmus SOMA. Tento algoritmus ve své základní podobě byl pro tuto problematiku jen velice obtížně použitelný. Nevyhovoval hned v několika bodech.

- Není efektivní prohledávat celou mapu, když nepřítel se z největší pravděpodobností vyskytuje na některé z mála pozic obsahující suroviny.
- Není vhodné nechávat bojovou jednotku jakožto jedince populace samotnou. Pokud by každý jedinec populace putoval vlastní cestou, byl by nepřítelem vyskytujícím se v koncentrovanější síle jednoduše eliminován.
- Z charakteristiky hry není možné zaručit náhodné rozložení populace po mapě. Každý hráč začíná na jedné startovní pozici. V průběhu hry sice nabývá na počtu budov, které dokáží produkovat bojové jednotky, ale za žádných okolností se nepodaří postavit tuto budovu kdekoli na mapě. Nepřítel by ji zničil. Taky není vhodné nechat bojovou, jednotu před zahrnutím jí do populace, poslat na náhodné místo, z důvodu jejího osamocení. Tabulka 4 znázorňuje parametry algoritmu SOMA implementovaného v botovi.

Parametr	Hodnota
PathLength	1
Step	0,5
PRT	0.8
D	2
PopSize	proměnlivá
Migrace	nekonečno
MinDiv	nespecifikováno

Tabulka 4: Parametry algoritmu SOMA

PathLength chci, aby jedinci doputovali přesně na místo leadera

Step je volena dostatečně velká hodnota parametru Step, aby jedinci doputovali co nejdál a přitom „nezabloudili“ na mapě. BWAPI rozhraní nabízí metodu „move“,

která nařídí jednotce dostat se na specifikované souřadnice. Jednotka si na to místo potom najde sama cestu. Pokud by se ale stalo, že by cílová souřadnice byla z důvodu charakteristiky mapy někde ve slepé uličce, tak by jednotka nejdříve navštívila tu, a až v dalším migračním kole by byla vygenerována nová souřadnice dále na cestě. Jedinec by se musel znovu vracet. Nebo v horším případě by se ze slepé uličky nemusel dostat vůbec.

PRT chci, aby jedinci byli jen málo pertrubovíni, neb směřovali přímo k cíli. Nepřítel bude buď tam, nebo jej jednotky potkají cestou. Pokud by hodnota *PRT* byla nastavena na malé číslo, vyskytl by se opět problém s roztříštěnou armádou jedinců, lehce zničitelnou koncentrovanou armádou nepřítele.

D optimalizuji pouze 2 parametry, a to jsou souřadnice na mapě x a y.

PopSize další ze zvláštností této implementace algoritmu je jeho proměnlivá velikost populace. Do populace jsou zahrnuty veškeré vyprodukované bojové jednotky. Ty samozřejmě časem přibývají a s bojem ubývají.

Migrace tento algoritmus je spuštěn paralelně po celou dobu běhu hry, proto je jeho hodnota nekonečno.

MinDiv tento parametr v této implementaci nebyl využit z důvodu jeho nekončícího běhu

6.1.1 Účelová funkce

Jako parametry výpočtu vhodnosti jsem zvolil vzdálenost bojové jednotky od pozic se surovinami, vzdálenost jednotky od startovních bodů na mapě, vzdálenost jednotky od některých strategicky významných pozic, např. základny nepřítele vyskytující se mimo surovinové bohatství, a součet životů nepřátelských jednotek kolem bojové jednotky v radiusu 100 jednotek na mapě. Velikost mapy může být třeba 4000×4000 jednotek. Díky tohoto posledního parametru musí SOMA optimalizovat dynamickou funkci, protože nepřátelské jednotky se pohybují, také vznikají a umírají. Rovnice 2 je rovnicí účelové funkce.

$$fitness = \sum_{i=0}^{m} hp_i w_i - \sum_{j=1}^{n} d_{Bj} w_j - \sum_{k=1}^{p} d_{Mk} w_k - \sum_{l=0}^{q} d_{IPl} w_l \tag{2}$$

m... počet jednotek v okruhu 100 od jedince

hp_i... počet životů nepřátelské jednotky v okruhu 100 od jedince

w_i... přidělená váha nepřátelské jednotky

n... počet startovních pozic

d_{Bj}... vzdálenost startovní pozice od jedince

w_j... přidělená váha startovní pozice (ze startu stejná pro všechny pozice (nepřítel může být na kterékoli z nich), po objevení nepřítele úprava váhy)

p... počet pozic se surovinami

d_{Mk}... vzdálenost jedince od pozice se surovinami

w_k... přidělená váha pozice se surovinami (může být proměnlivá v průběhu hry, podle výskytu nepřátel)

q... počet důležitých pozic na mapě (posledního střetu s nepřítelem, nějaké jiné strategicky významné pozice)

d_{IPl}... vzdálenost od důležitého bodu

w_l... přidělená váha důležité pozice

w_i	10
w_j	5
w_k	0,1
w_l	1

Tabulka 5: Nastavení vah

Tabulka 5 ukazuje váhy nastavené při simulacích. Největší váha je přidělena jednotkám. Ty totiž představují hrozbu a musí být eliminovány co nejdřív. Váha startovních pozic byla nastavena na číslo dostatečně velké oproti váhy pozice surovin, protože hlavním cílem hry je zničit nepřátelskou hlavní základnu, která se jistě nachází na startovní pozici a ne na pozici surovin. Tedy jistě že na startovních pozicích se suroviny nachází, ale ostatní pozice surovin slouží jako místo pro vedlejší základny tzv. *expanze*. Váha strategické pozice je nastavena na neutrální váhu. Tato hodnota nebyla testována. Tato váha by mohla být nastavená podle toho, o jaký typ pozice se jedná. Ať je to nepřátelská mobilní základna, nebo pozice poslední konfrontace s nepřítelem atd.

6.1.2 Populace

Jak již bylo řečeno, zvláštností této implementace algoritmu SOMA je jeho proměnlivá velikost populace (parametr *PopSize*). Bojové jednotky jsou brány jako členy populace v okamžiku jejich vytvoření a z populace odebrány, když jsou zabity. Toto ale není jediná zvláštnost populace. Celá populace je složena ze dvou, řekněme poddruhů. Jeden poddruh je klasická populace jednotek, pohybujících se po hyperploše. Tato populace je dynamická. Vedle ní obsahuje tato implementace ještě statický poddruh populace, který je výjimečný tím, že jedinci tohoto poddruhu se nehýbou. Tito jedinci jsou umístěny na všechny místa na mapě, kde se vyskytují suroviny a také do všech míst na mapě, které jsou startovními pozicemi hráčů. Tato populace v algoritmu přeskakuje fázi vlastní cesty a nemůže být nijak zničena. Může z ní ale být zvolen leader a po většinu času také leaderem bude jedinec tohoto poddruhu. Je to dáno tím, že podle účelové funkce na těchto všech pozicích se nacházejí všechny její statické lokální extrémy. Dynamické lokální extrémy (nepřátelské bojové jednotky) jsou nalezeny při pohybu dynamického poddruhu (bojových jednotek).

Tato implementace populace řeší všechny nevýhody použití klasické verze algoritmu SOMA řečené dříve. Jelikož jsou jedinci dynamického poddruhu pseudonáhodně rozestavěny (deterministicky rozmístěny podle znalosti pozic surovin a startovních pozic hráčů, náhodně z pohledu dynamického jedince populace, který téměř žádnou znalost prostředí nemá), není potřeba náhodně po mapě stavět budovy produkující bojové jednotky, ani není nutné je po jejich vyprodukování poslat na náhodné místo na mapě, kde by jistě padly.

6.1.3 Průběh chodu algoritmu na mapě pro dva hráče

Po startu algoritmu je jako leader zvolen statický jedinec, odpovídající druhé startovní pozici na mapě. Tam se nachází nepřátelská hlavní základna. Po vyprodukování jednoty se tato jednotka stane jedincem dynamické populace algoritmu SOMA a začíná jeho cesta přímo k leaderovi (hlavní nepřátelské základně). Může být zde implementována podmínka, že jeho cesta započne až s určitou velikostí dynamické populace, abychom dosáhli koncentrovanější síly. Na cestě může jedinec narazit na nepřátelské jednotky, což je dynamická složka účelové funkce, a pokusí se je zničit. Zde může být naprogramován bojový systém, třeba rozhodovací strom, který rozhodne, zda je síla mých jednotek dostatečná ke zničení nalezené armády nepřítele, nebo se dát raději na ústup a přednostně vyprodukovat další jednotky. Nebo taky hodnotící systém, v jakém pořadí by měly být zničeny nalezené nepřátelské jednotky podle velikosti hrozby, kterou představují. Dále tato implementace zajišťuje funkci tzv. hlídacího psa (watchdog), která se používá třeba v HW k restartování PC v případě zaseknutí (deadlocku). Zde by hrála funkci alarmu představujícího hrozbu nepřítele, útočícího na jednu z našich základen. Tento přínosný vedlejší efekt je způsoben tím, že statická populace obsahuje i jedince, představující pozici vlastní základny. Jelikož je váha nepřátelské jednotky nastavena na tak vysokou hodnotu, tak i když dynamická populace právě útočí na protivníkovu základnu a nepřítel ve

větším počtu útočí na tu naší, je za leadera zvolen jedinec statické populace, představující pozici vlastní základny, která je právě pod velkým útokem, tím pádem je právě útočící dynamická populace přitahována ke zdroji nepřátelského útoku.

6.1.4 Průběh chodu algoritmu na mapě pro více hráčů

Pro fungování takto implementované verze algoritmu SOMA na mapě s více startovními pozicemi by se musela doplnit ještě funkce, která by po nenalezení nepřítele na pozici leadera upravila jeho váhu na nulu. Tato váha by se mohla postupem času zvětšovat, až by dospěla do té míry, kdy by pozice byla opět zvolena za leadera, a tedy by donutila dynamické jedince místo znovu zkontrolovat.

6.2 Simulace

Rasa, která byla zvolena jako testovací a umělá inteligence byla speciálně vyvinuta na ni, je rasa Zergů. Výhodnost této rasy pro tyto účely vyplývá z její množstevní převahy, která je dána nízkou cenou jejich bojových jednotek. Jako testovací strategie byly zvoleny dvě. Jedna zahrnuje rychlý přímočarý útok směrem k nepříteli. Okamžitě po startu je vystavěna budova, která je nezbytná pro produkci prvního typu jednotek. Tato jednotka se jmenuje „Zergling“. Tuto jednotku ukazuje obr. 16 Jedná se o pozemní jednotku, která může útočit pouze na blízké cíle. Je velmi rychlá, ale má málo životů. Z jednoho vajíčka se vylíhnou dvě tyto jednotky. Má možnost dalšího vylepšení rychlosti a také se dokáže zahrabat, pokud je vyzkoumána tato zergská schopnost. Tato zmiňovaná strategie využívá možnosti zergské rasy vyprodukovat tyto jednotky v extrémně krátkém čase a zaskočit tak nepřítele neozbrojeného. Taktika se hodí na malé mapy pro dva hráče. Pokud by byla mapa větší, nebo možných nepřátelských startovních pozic více, tato strategie by nebyla efektivní, protože za dobu strávenou hledáním nepřítele, má onen nepřítel čas připravit dostatečnou obranu. Proti zkušeným lidským hráčům je tato strategie také málo efektivní, protože proti tomuto útoku se může hráč ubránit svými dělníky. Dále pokud selže prvních pár vln Zerglingů, je hráč odsouzen k nedostatku surovin a tedy lehkou kořistí pro hráče dosud bránícího. Tabulka 6 ukazuje úspěšnost této strategie na botech, kteří jsou ve hře již implementovaní. Hodnota vpravo je počet vítězství nepřítele.

Obrázek 16: Skupinka Zerglingů

Protivník	Skóre
Protoss	6 : 4
Terran	8 : 2
Zerg	9 : 1

Tabulka 6: Skóre simulací

Je potvrdila se výhoda rasy Protosů nad Zergama v začáteční fázi hry, kdy základní nepřátelská jednotka Protosů „Zealot“, která je schopna zabít až čtyři Zerglingy, pokud útočí postupně, a ne najednou. Zealot je jednotka útočící pouze na blízké cíle jako Zergling. Avšak disponuje větším počtem životů, štíty a uděluje větší poškození. 17 ukazuje tuto jednotku.

Obrázek 17: Skupinka Zealotů

Druhá strategie nebyla testována, z důvodu jejího nedokončení. Jedná se o strategii, která je ze začátku bitvy zaměřena na obranu. Princip spočívá v rychlém dosažení jednotky jménem „Lurker“. Jedná se o mutaci střelecké jednotky „Hydralisk“. Obr. 18 ukazuje obě tyto jednotky. Lurker je velice efektivní jednotka z důvodu, že k útoku musí být zahrabán. Tento efekt je podobný neviditelnosti. Zpočátku hy nepřátelé většinou nedisponují ničím, co by mohlo Lurkery odhalit. Tato jednotka útočí v podobě vlny bodců, které uberou životy všem nepřátelským jednotkám, které tyto bodce trefí. Tato strategie je vhodná pouze pro zkušenější hráče, kteří dokáží splnit všechny podmínky, které jsou potřeba k dosažení této jednotky, do doby, než zaútočí nepřítel. Tato strategie otevírá nekonečné možnosti dále taktizovat, protože hráč má dostatek surovin a je v relativním bezpečí před nepřítelem. V této strategii je dokončená do bodu postavení několika Lurkerů. Další vývoj hry je dokončen pouze do stereotypu, že se staví jednotka Hydralisk.

Videa ze simulací se nachází v příloze B.

Obrázek 18: Hydraliskové vlevo a Lurkeři vpravo

7 Závěr

Cílem této práce bylo implementovat bota do strategické hry StarCraft. Použití klasických technik pro umělou inteligenci se v tomto konkrétním případě osvědčilo. Rozhodování v implementaci měla na starosti technika „rozhodovací strom“ a to díky její jednoduchosti, a jednoduché rozšiřitelnosti. Rozhodovací strom produkoval akce, které byly po sléze zpracovány exekučním managerem. Ten spouštěl akce buď to paralelně ve vláknech, nebo sériově, po dokončení akce předchozí. Tento systém správy akcí má výhodu centralizovaného managementu všech vyprodukovaných akcí. Dále jako logika pohybu bojových jednotek byl zvolen algoritmus SOMA, který se po několika úpravách jevil jako velice efektivní řešení tohoto problému. Zajistil nám vedení útoku na nepřítele, a stejně tak plnil funkci alarmu, pokud byla naše základna nepřítelem napadena. Po jistých úpravách tedy jsou tyto evoluční techniky velice výhodným nástrojem v tomto segmentu.

Použití evolučního algoritmu SOMA nebylo zprvu to nejvhodnější řešení. Bojové jednotky by vyžadovaly prvotní náhodné rozmístění po mapě, což v tomto případě není příliš možné. Pro zvýšení efektivity byla upravena populace tohoto algoritmu. Kromě klasických pohybujících se (dynamických) jedinců dále obsahovala statické, nepohybující se jedince, kteří byli rozmístění do možných lokálních a globálních extrémů. Tento systém poddruhů populace rapidně zvedl použitelnost algoritmu na tomto konkrétním problému a představuje originální řešení optimalizačních problémů. Takové to řešení může být vhodné pro optimalizaci dynamické funkce, kde můžeme určit pozici některých statických lokálních extrémů.

Bot obsahuje nejdůležitější komponenty pro svoje fungování. Rozhodovací strom pro rozhodování, algoritmus SOMA pro logiku pohybu, centralizované zpracování akcí, rozhraní pro jednotlivé typy akcí. Z pohledu dalšího vylepšení takto implementovaného bota je zapotřebí rozšířit rozhodovací strom, popřípadě implementovat další pro různé strategie. Dále je vhodné vypracovat logiku bojové konfrontace s nepřítelem. Tím je myšleno strukturovaně vybírat cílové nepřátelské jednotky v pořadí podle vhodnosti jejich eliminace.

Tato práce se velice obtížně ladila z důvodu, že se laděné akce odehrávaly třeba i několik minut po začátku hry. Proto byl proces ladění velice zdlouhavý. Obecně považuji práci za úspěšnou, neboť díky ní bylo možno veškeré techniky odzkoušet v akci.

8 Reference

[1] BIGUS, Joseph P a Jennifer BIGUS. *Constructing intelligent agents using JAVA.* 2nd ed. New York: Wiley, 2001, xxii, 408 p. ISBN 047139601x.

[2] BUCKLAND, Mat. *Programming game AI by example.* Plano, Texas: Wordware Pub., 2005, xxi, 495 p. ISBN 1556220782.

[3] GREGORY, Jason. *Game engine architecture.* Natick: A K Peters, c2009, xx, 860 s. ISBN 9781568814131.

[4] KARTHIGAYAN, M., M. RIZON, R. NAGARAJAN a Sazali YAACOB. Genetic Algorithm and Neural Network for Face Emotion Recognition.*Affective Computing*. I-Tech Education and Publishing, 2008-05-01. DOI: 10.5772/6186. Dostupné z: `http://www.intechopen.com/books/affective_computing/genetic_algorithm_and_neural_network_for_face_emotion_recognition`

[5] MILLINGTON, Ian a John David FUNGE. *Artificial intelligence for games.* 2nd ed. Burlington: Morgan Kaufmann Publishers, c2009, xxiii, 870 s. ISBN 9780123747310.

[6] PALLMANN, David. *Programming bots, spiders, and intelligent agents in Microsoft Visual C++.* Redmond, Wash.: Microsoft Press, 1999, xvii, 679 p. ISBN 0735605653.

[7] POOLE, David L a Alan K MACKWORTH. *Artificial intelligence: foundations of computational agents.* New York: Cambridge University Press, 2010, xvii, 662 p. ISBN 9780521519007.

[8] WATSON, Mark. *Practical Artificial Intelligence Programming With Java.* November 11, 2008

[9] WISE, Edwin. *Hands-on AI with Java: smart gaming, robotics, and more.* New York: McGraw-Hill, 2005, p. cm. ISBN 0071424962.

[10] Zelinka, Ivan. *Evoluční výpočetní techniky: principy a aplikace.* 1. české vyd. Praha: BEN, 2009, 534 s. ISBN 978-80-7300-218-3.

Přílohy

Příloha A: Obrázky ze hry

A.1 Protoská základna

Obrázek 19: Protoská základna 1

Obrázek 20: Protoská základna 2

A.2 Terranská základna

Obrázek 21: Terranská základna 1

Obrázek 22: Terranská základna 2

A.3 Zergská základna

Obrázek 23: Zergská základna 1

Obrázek 24: Zergská základna 2

A.4 Další obrázky ze hry

Obrázek 25: Protoss napadený Terranem (dominující letecká jednotka se jmenuje Battlecruiser)

Obrázek 26: Ukázka síly Protosů (průhledné jednotky jsou neviditelné)

Obrázek 27: Terranská vzdušná jednotka Battlecruiser používá schopnost Yamato cannon

Obrázek 28: Protoské kouzlo Psionic Storm proti mase zergských jednotek

Obrázek 29: některé zergské jednotky (zleva: Ultralisk, Queen, Defiler; nahoře Overlord)

Obrázek 30: Zergské vzdušné síly (zleva: Mutalisk, Guardian, Devourer; nahoře: Scourge)

Příloha B: Příloha na CD/DVD

Obsah CD/DVD:

- StarCraft - složka s projektem
- README.txt
- simulace - složka s videy chování bota
- digitální podoba bakalářské práce